Universidad Técnica Federico Santa María

DEPARTAMENTO DE INGENIERÍA ELECTRONICA

ELO 325 - Seminario de Computadores I

Guía 1

Estudiante **ROL**
Juan Aguilera 201621028-8

Paralelo: **1**

Profesor

Fernando Auat

Ayudante

Nicolás Aguayo

Fecha : 7 de noviembre de 2021

Índice

Índice de figuras

Índice de cuadros

1. Información adicional

Para el desarrollo de esta guía se programo exclusivamente usando `Python`, adicionalmente para la reproducción de los códigos presentes en **Anexos** se debe tener en cuenta que los programas fueron desarrollados en el entorno de desarrollo de *Spyder*, así como en en *VS Code*, instalando las siguientes bibliotecas:

- numpy
- matplotlib
- keyboard
- sim

Todos los códigos y el proyecto en `CoppeliaSim` también se encuentran en este enlace.

2. Ejercicio 1

El desarrollo de este ejercicio se encuentra resuelto en el anexo **Pregunta 1**, donde en dicho código se utilizan los siguientes parámetros:

- **R = 0.225** corresponde al radio del robot en metros.
- **dT = 10** corresponde al tiempo de muestreo usado para discretear las ecuaciones en segundos.

2.1.

Se generaron los códigos necesarios para formar un vector de puntos para las coordenadas `x` e `y`, en particular tenemos:

Para el camino circular se utiliza el siguiente código:

```
for i in range(pts + 1):
    x.append(2*R1*i/pts)
for i in range(pts+1):
    y.append(np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts,-1,-1):
    y.append(-np.sqrt(R1**2-(x[i]-R1)**2))
x = x + x[::-1]
```

Donde `pts` es una constante que representa la cantidad de puntos necesaria para generar media circunferencia, `R1` es el radio de la circunferencia a generar y finalmente `x` e `y` son los vectores de las posiciones. El código completo para generar y graficar el camino circular se muestra en el anexo Camino Circular.

Para el camino cuadrado se utiliza el siguiente código:

```
for i in range(L+1):
    x.append(D*i/L)
    y.append(0)
for i in range(L+1):
    y.append(D*i/L)
    x.append(D)
for i in range(L,-1,-1):
    x.append(D*i/L)
    y.append(D)
for i in range(L,-1,-1):
    y.append(D*i/L)
    x.append(0)
```

Donde `L` es una constante que representa la cantidad de puntos necesaria para generar cada lado del cuadrado, `D` es largo de cada linea a generar. `x, y` son los vectores de las posiciones. El código completo para generar y graficar el camino cuadrado se muestra en el anexo Camino Cuadrado.

Para el camino 8-invertido se utiliza el siguiente código:

```
for i in range(2*pts + 2):
    x.append(2*R1*i/pts)
```

```
for i in range(pts+1):
    y.append(np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts+1):
    y.append(-np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts,-1,-1):
    y.append(np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts,-1,-1):
    y.append(-np.sqrt(R1**2-(x[i]-R1)**2))

x = x + x[::-1]
```

Donde `pts` es una constante que representa la cantidad de puntos necesaria para generar media circunferencia, `R1` es el radio de las circunferencias que generan la forma de 8 y finalmente `x` e `y` son los vectores de las posiciones. El código completo para generar y graficar el camino 8-invertido se muestra en el anexo Camino 8-invertido.
Los recorridos graficados se muestran en la figura 1.

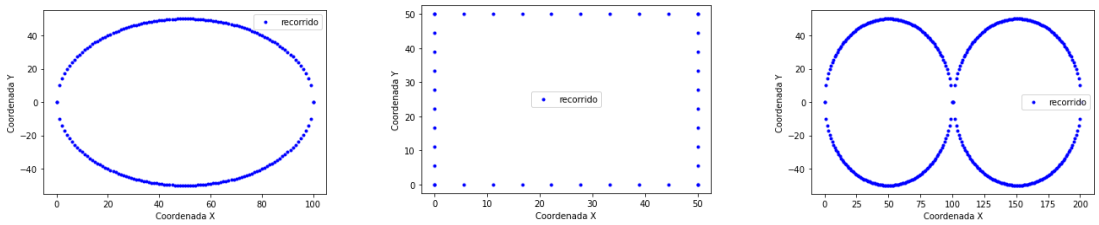

Figura 1: Grafica de recorridos

2.2.

Se generaron los códigos necesarios para formar tres vectores que representan las velocidades v_1, v_2, v_3. Para todos los caminos se utiliza el mismo código, mostrado a continuación:

```
A = (2/3)* np.array([[-np.sin(th_1), -np.sin(th_1+th_2), -np.
    sin(th_1+th_3)],
                     [np.cos(th_1), np.cos(th_1+th_2), np.cos(
    th_1+th_3)],
                     [0.5/R, 0.5/R, 0.5/R]])
A_inv = np.linalg.inv(A)

for i in range(x.size-1):
     V.append(np.array( np.matmul( (A_inv/dT) , np.transpose(np
    .array([x[i+1], y[i+1], 0]) - np.array([ x[i],y[i], 0])) ))
    )
V.append(np.array([0,0,0]))
```

Donde `th_1, th_2 y th_3` corresponden a constantes que representan ángulos en radianes, equivalentes a 30, 120, 240 grados respectivamente. `A` corresponde a la matriz de transformación para el modelo cinemático directo del robot omidireccional con ruedas suecas visto en clases, `A_inv` corresponde a la inversa de dicha matriz y finalmente `V` corresponde a una matriz cuyas columnas son los vectores de velocidad buscados.

El código busca representar la siguiente ecuación discretisada:

$$\begin{bmatrix} x \\ y \\ \theta_r \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ y \\ \theta_r \end{bmatrix}_{k} + \Delta t \frac{2}{3} \begin{bmatrix} -sin\theta_1 & -sin(\theta_1+\theta_2) & -sin(\theta_1+\theta_3) \\ cos\theta_1 & cos(\theta_1+\theta_2) & cos(\theta_1+\theta_3) \\ \frac{1}{2R} & \frac{1}{2R} & \frac{1}{2R} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad (1)$$

La cual por simplicidad expresaremos de la siguiente manera:

$$X_{k+1} = X_k + \Delta t \cdot A \cdot V$$

la cual despejamos para obtener los vectores de velocidad buscados:

$$V = X_k + \frac{A^*}{\Delta t}[X_{k+1} - X_k]$$

Donde A^* corresponde a la pseudo inversa de `A`.

La generación de los perfiles de velocidad para cada uno de los caminos deseados se muestra en los anexos Camino Circular, Camino Cuadrado y Camino 8-invertido.

2.3.

Para obtener las coordenadas, en los 3 tipos de camino, que necesita el robot para moverse a partir de las velocidades generadas en el punto anterior, se utiliza el siguiente código:

```
X0 = np.array([0,0,(1/6)*np.pi])
X1 = []

X1.append(X0)
for i in range(x.size-1):
    X1.append(X0 + dT*( np.matmul(A,np.transpose(np.array([V1[
    i],V2[i],V3[i]])))  ))
    X0 = X1[-1]
```

El cual representa el uso directo de la ecuación 1, donde `X0` representa el vector de posición en x_k y `X1` es un una matriz cuyas filas son los vectores de posición de cada instante k.

Para generar una animación del robot moviéndose a partir de las coordenadas resultantes de las velocidades obtenidas, se utiliza el siguiente código:

```
fig, ax = plt.subplots()
ax.set_xlim(0,4*R1)
ax.set_ylim(-R1,R1)
line, = ax.plot(0,0)

def animation_frame(i):
    x_data.append(x_new[i])
    y_data.append(y_new[i])

    line.set_xdata(x_data)
    line.set_ydata(y_data)
    return line,

animation = FuncAnimation(fig, func=animation_frame, frames=np
    .arange(x.size),interval=dT)
plt.show()
```

Donde se dibuja el movimiento del robot de cada punto al siguiente cada Δt, representando así el movimiento del robot.

En la figura 2 se muestran los gráfico de velocidad vs tiempo para cada uno de los caminos.

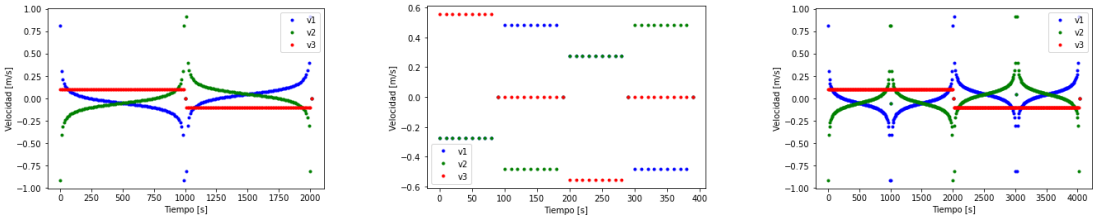

Figura 2: Grafica de velocidad vs tiempo.

Este gráfico se puede interpretar gracias a la figura 3, donde se muestra los vectores de velocidad en cada rueda del robot respecto a las coordenadas locales $< r >$ y globales $< G >$.

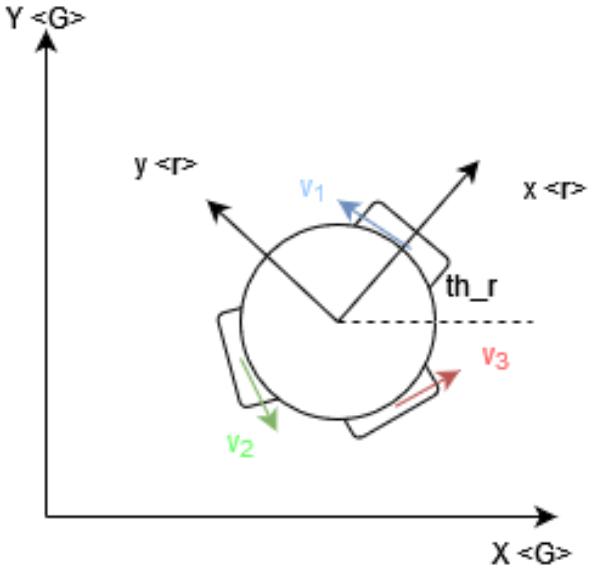

Figura 3: Esquema de velocidades para robot omnidireccional.

2.4.

En este caso, el ángulo inicial para θ_1 es de 0 grados, a diferencia de los puntos anteriores donde se mantenía en 30 grados.

Para imitar el movimiento de un uniciclo definimos el norte como la coordenada $x < r >$ mostrada en la figura 3, el cual debe tener la dirección al movimiento, es decir, la suma de los vectores de velocidad v_1, v_2 y v_3 tiene la misma dirección que $x < r >$. Para esto generamos un nuevo vector de coordenadas de recorrido, el cual se encarga además de generar los ángulos necesarios para girar en cada esquina, esto se genera con el siguiente código:

```
1 for i in range(L+1):
2     x.append(D*i/L)
```

```
    y.append(0)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi/2)*(i/G)) # llega a 90 grados
    x.append(x[-1])
    y.append(y[-1])
for i in range(L+1):
    y.append(D*i/L)
    x.append(D)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi/2)+ (np.pi/2)*(i/G)) # llega a 180 grados
    x.append(x[-1])
    y.append(y[-1])
for i in range(L,-1,-1):
    x.append(D*i/L)
    y.append(D)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi)+(np.pi/2)*(i/G)) # llega a -90 grados
    x.append(x[-1])
    y.append(y[-1])
for i in range(L,-1,-1):
    y.append(D*i/L)
    x.append(0)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi+np.pi/2)+(np.pi/2)*(i/G)) # llega a 0
   grados
    x.append(x[-1])
    y.append(y[-1])
```

Cuyos parámetros son los mismos para el camino cuadrado visto anteriormente, así como el perfil de velocidades se genera de la misma forma que visto anteriormente.

El gráfico de velocidades vs tiempo se muestra en la figura 4.

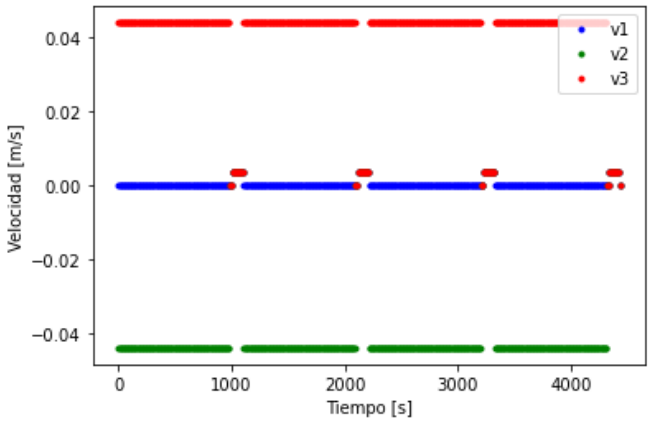

Figura 4: Velocidades vs tiempo para movimiento como uniciclo.

Donde notamos que la rueda que representa el norte del robot, esta con velocidad cero en todo momento, salvo cuando el robot debe girar para cambiar su ángulo.

En el anexo Robot omnidireccional como uniciclo se comprueba que utilizando el modelo cinemático del uniciclo se llega al mismo resultado.

3. Ejercicio 2

3.1.

Para este ejercicio, se elige entre los modelos disponibles en `CoppeliaSim` el vehículo *car-like* `Manta` el cual se encuentra en *model browser* → *vehicles* → *manta with differential.ttm*.

Para el sensor LiDAR, se elige el `Hakuyo Fast`, el cual se encuentra dentro de `CoppeliaSim` en *model browser* → *components* → *sensors* → *Hokuyo URG 04LX UG1_Fast.ttm*.

Luego, se utiliza la ventana *Scene hierarchy* dentro de `CoppeliaSim`, se arrastra el sensor `Hakuyo Fast` dentro de los componentes de `Manta`, de esta forma obteniendo un robot *car-like* con sensor LiDAR incorporado.

El resultado de esto, junto a la incorporación de más elementos a la escena, se muestra en la figura 5.

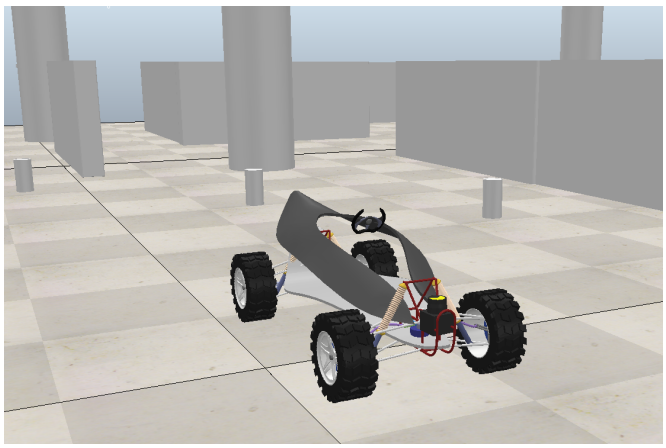

Figura 5: Robot car-like con sensor LiDAR.

3.2.

Para este ejercicio, es necesario conectar de forma remota `CoppeliaSim` a nuestra API de *Python*, para lo cual se usa la siguiente función:

```
1 def connect(port):
2     sim.simxFinish(-1)
3     clientID=sim.simxStart('127.0.0.1', port, True, True,
    2000, 5)
4     if clientID == 0: print("conectado a", port)
5     else: print("no se pudo conectar")
6     return clientID
```

Luego, se agregar la siguiente linea al *script* del objeto al cual se requiere controlar (en este caso a *Manta*) mediante la API:

```
1 simRemoteApi.start(19999)
```

La cual agregamos dentro de *sysCall_init()* en *Child script (Manta)*, en particular, borramos todo el *script* anterior, solo dejando la conexión a la API como se muestra en el anexo **Manta Child script**.

Luego, se advierte que el *script* original que viene con *Manta* utiliza funciones que controlan la velocidad de su moto, utilizando estas funciones y elementos disponibles se genero la API mostrada en el anexo **Movimiento de `Car-Like Robot` usando el teclado** que permite controlar el vehículo utilizando el teclado. El modo de control se describe a continuación:

- **w:** aumenta la velocidad del vehículo en una unidad, llamada dV.
- **s:** disminuye la velocidad en dV.
- **a:** aumenta el ángulo de giro del vehículo, lo que le permite doblar a la izquierda.
- **d:** reduce el ángulo de giro del vehículo, lo que le permite doblar a la derecha.
- **espacio:** disminuye la velocidad absoluta hasta dejarla en cero.

Además, cuando no se esta presionando ninguna tecla, la velocidad del vehículo se reduce hasta detenerse, simulado el roce.

3.3.

El primer acercamiento para este problema es obtener los datos del sensor *fastHokuyo*, lo cual en primer lugar se logra modificando un poco el *Child script (fastHokuyo)* agregando las lineas donde se comenta *added* como se muestra en el anexo Child Script fastHokuyo. La idea de esto es poder recibir un arreglo con todas las coordenadas en Y donde se detecta un obstáculo.

Luego se obtienen los primeros gráficos graficando X, Y segun el arreglo de datos en Y entregados por el sensor, con el siguiente código:

```
1  #primeros datos no validos
2 rC, ranges = sim.simxGetStringSignal(clientID, 'scan ranges',
     sim.simx_opmode_streaming)
3 time.sleep(0.1)
4
5 while(1):
6     time.sleep(0.1)
7     # Obtenga datos v lidos
8     rC, ranges = sim.simxGetStringSignal(clientID, 'scan
     ranges', sim.simx_opmode_buffer)
9     #Convertir cadena a lista flotante, el valor en la lista
     es el valor medido del radar
10    ranges = sim.simxUnpackFloats(ranges) sim.
     simx_opmode_buffer)
11    x = range(len(ranges))
12    y = ranges
13    # Dibuja el resultado
14    plt.scatter(x,y)
15    plt.show()
```

Luego, asociando estos datos al código utilizado para mover el robot con el teclado, podemos obtener un gráfico de los objetos frente al robot a medida que este se mueve. Este código se muestra en el anexo Uso del sensor Hokuyo.

Para obtener la ubicación actual del robot, se usa el componente *GPS.tmm* disponible en `CoppeliaSim`. La idea de obtener los datos de este sensor, es que

al iniciar se obtengan las coordenadas iniciales en el punto exacto donde se ubica dentro del robot (en este caso justo en la ubicación del sensor LiDAR). Para probar este concepto usamos el código siguiente:

```
returnCode, gps_handler = sim.simxGetObjectHandle(clientID,'
    GPS', sim.simx_opmode_blocking)
returnCode, gps_handler = sim.simxGetObjectHandle(clientID,'
    GPS', sim.simx_opmode_blocking)
```

Donde *pos* corresponde al vector de coordenadas globales x, y, z del componente.

También es necesario conocer el ángulo en el θ_r entre $x_<r >$ y $x_<G >$, con esta idea se intenta usar el componente *GyroSensor*.

Finalmente se podría construir el mapa del entorno obteniendo las mediciones del sensor LiDAR junto a la información del *GPS* y el *GyroSensor*.

Los resultados obtenidos se muestran en la figura 6, donde se logra apreciar la correcta obtención de los datos.

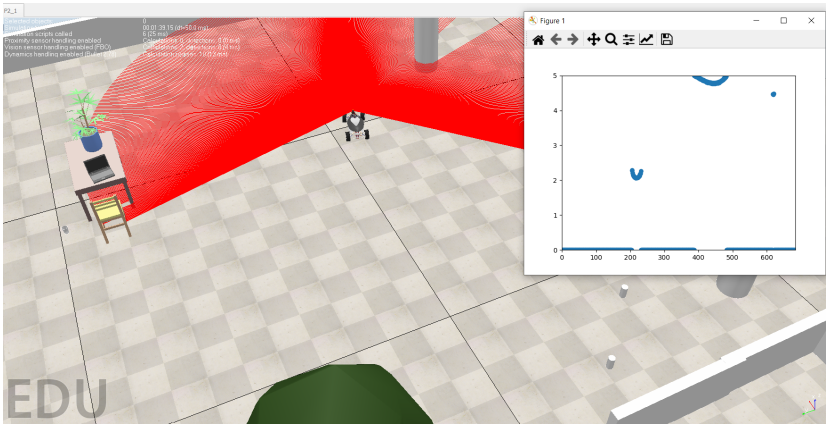

Figura 6: Gráfico obtenido por sensor LiDAR.

4. Anexos

4.1. Pregunta 1

4.1.1. Camino Circular

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  5 19:07:48 2021

@author: juan_
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation


# Camino circular
R1 = 50 # radio de la circunferencia  [m]
pts = 100 # puntos necesarios para una semi-circunferencia.
x = []
y = []
for i in range(pts + 1):
    x.append(2*R1*i/pts)
for i in range(pts+1):
    y.append(np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts,-1,-1):
    y.append(-np.sqrt(R1**2-(x[i]-R1)**2))

x = x + x[::-1]
x = np.array(x) # arreglo de puntos de la coordenada x
y = np.array(y) # arreglo de puntos de la coordenada y

plt.plot(x, y, '.', color='blue',label="recorrido")
plt.xlabel("Coordenada X")
plt.ylabel("Coordenada Y")
plt.legend()
plt.show()

## 1.2 perfiles de velocidad

R = 0.225 # radio del robot [m]
# perfil de velocidades de circunferencia
#defici n de los angulos en rad (se mantendran constante)
th_1 = (1/6)*np.pi #30 grados  # theta_1 = theta_r
th_2 = (2/3)*np.pi  # 120 grados
th_3 = (4/3)*np.pi  # 240 grados

# el robot va a manter su angulo en todo momento

dT = 10 #delta tiempo [s]

A = (2/3)* np.array([[-np.sin(th_1), -np.sin(th_1+th_2), -np.
    sin(th_1+th_3)],
                     [np.cos(th_1), np.cos(th_1+th_2), np.cos(
    th_1+th_3)],
                     [0.5/R, 0.5/R, 0.5/R]]) # matriz de
    transformaci n
```

```
A_inv = np.linalg.inv(A)

V = [] # matriz de velocidades

# obs: se mantiene los angulos theta_k y thera_k+1 en cero, ya que estos no cambian en ninguna iteracion
for i in range(x.size-1):
    V.append(np.array( np.matmul( (A_inv/dT) , np.transpose(np.array([x[i+1], y[i+1], 0]) - np.array([ x[i],y[i], 0])) ) ))
V.append(np.array([0,0,0]))

V = np.array(V)

# separamos por vectores de velocidad [v1,v2,v3]
V1 = np.array(V[:,0])
V2 = np.array(V[:,1])
V3 = np.array(V[:,2])

## 1.3 re hacer el circulo

#Circular
#usaremos la misma matriz A calculada antes, y el mismo delta T

X0 = np.array([0,0,(1/6)*np.pi]) # vector de coordenadas en k
# seteamos la primera posicion, en el origen
X1 = [] # vector de coordenadas en k + 1

X1.append(X0) # seteamos la primera posicion
for i in range(x.size-1):
    X1.append(X0 + dT*( np.matmul(A,np.transpose(np.array([V1[i],V2[i],V3[i]])))))
    X0 = X1[-1]

X1 = np.array(X1)

x_new = []
y_new = []
th = []

x_new = np.array(X1[:,0])
y_new = np.array(X1[:,1])
th = np.array(X1[:,2])

## animaci n
x_data = []
y_data = []

fig, ax = plt.subplots()
ax.set_xlim(0,2*R1)
ax.set_ylim(-R1,R1)
line, = ax.plot(0,0)

def animation_frame(i):
    x_data.append(x_new[i])
    y_data.append(y_new[i])

    line.set_xdata(x_data)
    line.set_ydata(y_data)
```

```
105     return line,
106 
107 animation = FuncAnimation(fig, func=animation_frame, frames=np
        .arange(x.size),interval=dT)
108 plt.show()
109 
110 plt.plot(x_new, y_new, '.', color='blue')
111 plt.show()
112 
113 v_time = [] # tiempo para graficar las velocidades
114 ## grafico de la se al de control vs tiempo
115 for i in range(x.size):
116     v_time.append(dT*i)
117 v_time = np.array(v_time)
118 
119 plt.plot(v_time,V1, '.', color='blue',label="v1")
120 plt.plot(v_time,V2, '.', color='green',label="v2")
121 plt.plot(v_time,V3, '.', color='red',label="v3")
122 plt.xlabel("Tiempo [s]")
123 plt.ylabel("Velocidad [m/s]")
124 plt.legend()
125 plt.show()
```

4.1.2. Camino Cuadrado

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  5 19:07:48 2021

@author: juan_
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# camino cuadrado
L = 10 # largo de cada lado, en puntos
D = 50 # distancia recorrida por cada lado [m]

x = []
y = []
L = L-1 # para poder ingresar el valor de puntos exacto
for i in range(L+1):
    x.append(D*i/L)
    y.append(0)
for i in range(L+1):
    y.append(D*i/L)
    x.append(D)
for i in range(L,-1,-1):
    x.append(D*i/L)
    y.append(D)
for i in range(L,-1,-1):
    y.append(D*i/L)
    x.append(0)

x = np.array(x) # arreglo de puntos de la coordenada x
y = np.array(y) # arreglo de puntos de la coordenada y

plt.plot(x, y, '.', color='blue',label="recorrido")
plt.xlabel("Coordenada X")
plt.ylabel("Coordenada Y")
plt.legend()
plt.show()
## 1.2 perfiles de velocidad

R = 0.225 # radio del robot [m]
# perfil de velocidades de circunferencia
#defici n de los angulos en rad (se mantendran constante)
th_1 = (1/6)*np.pi #30 grados  # theta_1 = theta_r
th_2 = (2/3)*np.pi  # 120 grados
th_3 = (4/3)*np.pi  # 240 grados

# el robot va a manter su angulo en todo momento, por lo que
    theta siempre se mantendra en cero

dT = 10 #delta tiempo [s]

A = (2/3)* np.array([[-np.sin(th_1), -np.sin(th_1+th_2), -np.
    sin(th_1+th_3)],
                     [np.cos(th_1), np.cos(th_1+th_2), np.cos(
    th_1+th_3)],
```

```
                    [0.5/R, 0.5/R, 0.5/R]]) # matriz de transformaci n
A_inv = np.linalg.inv(A)

V = [] # matriz de velocidades

for i in range(x.size-1):
    V.append(np.array( np.matmul( (A_inv/dT) , np.transpose(np.array([x[i+1], y[i+1], 0]) - np.array([ x[i],y[i], 0])) ) ))
V.append(np.array([0,0,0]))

V = np.array(V)

# separamos por vectores de velocidad [v1,v2,v3]
V1 = np.array(V[:,0])
V2 = np.array(V[:,1])
V3 = np.array(V[:,2])

## 1.3

#Circular
#usaremos la misma matriz A calculada antes, y el mismo delta T

X0 = np.array([0,0,(1/6)*np.pi]) # vector de coordenadas en k
    # seteamos la primera posicion, en el origen
X1 = [] # vector de coordenadas en k + 1

X1.append(X0) # seteamos la primera posicion
for i in range(x.size-1):
    X1.append(X0 + dT*( np.matmul(A,np.transpose(np.array([V1[i],V2[i],V3[i]])))))
    X0 = X1[-1]

X1 = np.array(X1)

x_new = []
y_new = []
th = []

x_new = np.array(X1[:,0])
y_new = np.array(X1[:,1])
th = np.array(X1[:,2])


## animaci n

x_data = []
y_data = []

fig, ax = plt.subplots()
ax.set_xlim(-10,D+10)
ax.set_ylim(-10,D+10)
line, = ax.plot(0,0)

def animation_frame(i):
    x_data.append(x_new[i])
    y_data.append(y_new[i])

```

```
109     line.set_xdata(x_data)
110     line.set_ydata(y_data)
111     return line,
112 
113 animation = FuncAnimation(fig, func=animation_frame, frames=np
        .arange(x.size),interval=dT)
114 plt.show()
115 
116 plt.plot(x_new, y_new, '.', color='blue')
117 plt.show()
118 
119 v_time = [] # tiempo para graficar las velocidades
120 ## grafico de la se al de control vs tiempo
121 for i in range(x.size):
122     v_time.append(dT*i)
123 v_time = np.array(v_time)
124 
125 plt.plot(v_time,V1, '.', color='blue',label="v1")
126 plt.plot(v_time,V2, '.', color='green',label="v2")
127 plt.plot(v_time,V3, '.', color='red',label="v3")
128 plt.xlabel("Tiempo [s]")
129 plt.ylabel("Velocidad [m/s]")
130 plt.legend()
131 plt.show()
```

4.1.3. Camino 8-invertido

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  5 19:07:48 2021

@author: juan_
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#camino 8-invertido
# camino 8-invertido

R1 = 50 # radio de la circunferencia para cada lobulo
pts = 100 # puntos necesarios para una semi-circunferencia.
x = []
y = []
for i in range(2*pts + 2):
    x.append(2*R1*i/pts)
for i in range(pts+1):
    y.append(np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts+1):
    y.append(-np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts,-1,-1):
    y.append(np.sqrt(R1**2-(x[i]-R1)**2))
for i in range(pts,-1,-1):
    y.append(-np.sqrt(R1**2-(x[i]-R1)**2))

x = x + x[::-1]
x = np.array(x) # arreglo de puntos de la coordenada x
y = np.array(y) # arreglo de puntos de la coordenada y

plt.plot(x, y, '.', color='blue',label="recorrido")
plt.xlabel("Coordenada X")
plt.ylabel("Coordenada Y")
plt.legend()
plt.show()

## 1.2 perfiles de velocidad

R = 0.225 # radio del robot
# perfil de velocidades de circunferencia
#defici n de los angulos en rad (se mantendran constante)
th_1 = (1/6)*np.pi #30 grados  # theta_1 = theta_r
th_2 = (2/3)*np.pi  # 120 grados
th_3 = (4/3)*np.pi  # 240 grados

# el robot va a manter su angulo en todo momento, por lo que
    theta siempre se mantendra en cero

dT = 10 #delta tiempo, [s]

A = (2/3)* np.array([[-np.sin(th_1), -np.sin(th_1+th_2), -np.
    sin(th_1+th_3)],
                     [np.cos(th_1), np.cos(th_1+th_2), np.cos(
    th_1+th_3)],
```

```
                    [0.5/R, 0.5/R, 0.5/R]]) # matriz de transformaci n
A_inv = np.linalg.inv(A)

V = [] # matriz de velocidades

for i in range(x.size-1):
    V.append(np.array( np.matmul( (A_inv/dT) , np.transpose(np.array([x[i+1], y[i+1], 0]) - np.array([ x[i],y[i], 0])) ) ))
V.append(np.array([0,0,0]))

V = np.array(V)

# separamos por vectores de velocidad [v1,v2,v3]
V1 = np.array(V[:,0])
V2 = np.array(V[:,1])
V3 = np.array(V[:,2])

## 1.3 8-invertido

#Circular
#usaremos la misma matriz A calculada antes, y el mismo delta T

X0 = np.array([0,0,(1/6)*np.pi]) # vector de coordenadas en k
    # seteamos la primera posicion, en el origen
X1 = [] # vector de coordenadas en k + 1

X1.append(X0) # seteamos la primera posicion
for i in range(x.size-1):
    X1.append(X0 + dT*( np.matmul(A,np.transpose(np.array([V1[i],V2[i],V3[i]])))  ))
    X0 = X1[-1]

X1 = np.array(X1)

x_new = []
y_new = []
th = []

x_new = np.array(X1[:,0])
y_new = np.array(X1[:,1])
th = np.array(X1[:,2])


## animaci n

x_data = []
y_data = []

fig, ax = plt.subplots()
ax.set_xlim(0,4*R1)
ax.set_ylim(-R1,R1)
line, = ax.plot(0,0)

def animation_frame(i):
    x_data.append(x_new[i])
    y_data.append(y_new[i])

```

```
109     line.set_xdata(x_data)
110     line.set_ydata(y_data)
111     return line,
112 
113 animation = FuncAnimation(fig, func=animation_frame, frames=np
        .arange(x.size),interval=dT)
114 plt.show()
115 
116 plt.plot(x_new, y_new, '.', color='blue')
117 plt.show()
118 
119 v_time = [] # tiempo para graficar las velocidades
120 ## grafico de la se al de control vs tiempo
121 for i in range(x.size):
122     v_time.append(dT*i)
123 v_time = np.array(v_time)
124 
125 plt.plot(v_time,V1, '.', color='blue',label="v1")
126 plt.plot(v_time,V2, '.', color='green',label="v2")
127 plt.plot(v_time,V3, '.', color='red',label="v3")
128 plt.xlabel("Tiempo [s]")
129 plt.ylabel("Velocidad [m/s]")
130 plt.legend()
131 plt.show()
```

4.1.4. Robot omnidireccional como uniciclo

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  6 16:27:15 2021

@author: juan_
"""

## P1.4

## implementaci n de movimiento como uniciclo

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# camino cuadrado
L = 100 # largo de cada lado, en puntos
D = 50 # distancia recorrida por cada lado [m]
G = 10 # tiempo de giro [en puntos]

x = []
y = []
th = []

th.append(0) # angulo theta_r inicial
L = L-1 # para poder ingresar el valor de puntos exacto
for i in range(L+1):
    x.append(D*i/L)
    y.append(0)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi/2)*(i/G)) # llega a 90 grados
    x.append(x[-1])
    y.append(y[-1])
for i in range(L+1):
    y.append(D*i/L)
    x.append(D)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi/2)+ (np.pi/2)*(i/G)) # llega a 180 grados
    x.append(x[-1])
    y.append(y[-1])
for i in range(L,-1,-1):
    x.append(D*i/L)
    y.append(D)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi)+(np.pi/2)*(i/G)) # llega a -90 grados
    x.append(x[-1])
    y.append(y[-1])
for i in range(L,-1,-1):
    y.append(D*i/L)
    x.append(0)
    th.append(th[-1])
for i in range(G+1):
    th.append((np.pi+np.pi/2)+(np.pi/2)*(i/G)) # llega a 0
   grados
```

```
57      x.append(x[-1])
58      y.append(y[-1])
59
60  x = np.array(x) # arreglo de puntos de la coordenada x
61  y = np.array(y) # arreglo de puntos de la coordenada y
62  th = np.array(th)# arreglo de puntos el angulo theta_r
63
64  plt.plot(x, y, '.', color='blue')
65  plt.show()
66
67  ## 1.4 perfiles de velocidad
68
69  R = 0.225 # radio del robot [m]
70  # perfil de velocidades para el recorrido cuadrado
71  #defici n de los angulos en rad (se mantendran constante)
72  th_2 = (2/3)*np.pi  # 120 grados
73  th_3 = (4/3)*np.pi  # 240 grados
74
75  # el robot va a manter su angulo en todo momento, por lo que
        theta siempre se mantendra en cero
76
77  dT = 10 #delta tiempo, en segundos
78  A = []
79  for i in range(x.size):
80      A.append( (2/3)* np.array([[-np.sin(th[i]), -np.sin(th[i]+
        th_2), -np.sin(th[i]+th_3)],
81                          [np.cos(th[i]), np.cos(th[i]+th_2), np.
        cos(th[i]+th_3)],
82                          [0.5/R, 0.5/R, 0.5/R]]) ) # matriz de
        transformaci n
83  A = np.array(A)
84
85  V = [] # matriz de velocidades
86
87  for i in range(x.size-1):
88      V.append(np.array( np.matmul( (np.linalg.inv(A[i])/dT) ,
        np.transpose(np.array([x[i+1], y[i+1], th[i+1]]) - np.array
        ([ x[i],y[i],th[i]])) )))
89  V.append(np.array([0,0,0]))
90
91  V = np.array(V)
92
93  # separamos por vectores de velocidad [v1,v2,v3]
94  V1 = np.array(V[:,0])
95  V2 = np.array(V[:,1])
96  V3 = np.array(V[:,2])
97
98  ## 1.3
99  #usaremos la misma matriz A calculada antes, y el mismo delta
        T
100
101 X0 = np.array([0,0,0]) # vector de coordenadas en k # seteamos
        la primera posicion, en el origen
102 X1 = [] # vector de coordenadas en k + 1
103
104 X1.append(X0) # seteamos la primera posicion
105 for i in range(x.size-1):
106     X1.append(X0 + dT*( np.matmul(A[i],np.transpose(np.array([
        V1[i],V2[i],V3[i]])))) )
107     X0 = X1[-1]
```

```
108
109 X1 = np.array(X1)
110
111 x_new = []
112 y_new = []
113 th_new = []
114
115 x_new = np.array(X1[:,0])
116 y_new = np.array(X1[:,1])
117 th_new = np.array(X1[:,2])
118
119
120 ## animaci n
121
122 x_data = []
123 y_data = []
124
125 fig, ax = plt.subplots()
126 ax.set_xlim(-10,D+10)
127 ax.set_ylim(-10,D+10)
128 line, = ax.plot(0,0)
129
130 def animation_frame(i):
131     x_data.append(x_new[i])
132     y_data.append(y_new[i])
133
134     line.set_xdata(x_data)
135     line.set_ydata(y_data)
136     return line,
137
138 animation = FuncAnimation(fig, func=animation_frame, frames=np
        .arange(x.size),interval=dT)
139 plt.show()
140
141 plt.plot(x_new, y_new, '.', color='blue')
142 plt.show()
143
144
145 ########################################
146 ########################################
147 ## implementaci n uniciclo para comprobar
148
149
150 ## 1.4 perfiles de velocidad para uniciclo
151
152 # el robot va a manter su angulo en todo momento, por lo que
        theta siempre se mantendra en cero
153
154 dT = 10 #delta tiempo, un misterio
155 Au = []
156 for i in range(x.size):
157     Au.append(np.array([[np.cos(th[i]), 0],
158                         [np.sin(th[i]), 0],
159                         [0, 1]] )) # matriz de transformaci n
        para el unicilo
160 Au = np.array(Au)
161
162 Vu = [] # matriz de velocidades [v,w] uniciclo
163
164 for i in range(x.size-1):
```

```
    Vu.append(np.array( np.matmul( (np.linalg.pinv(Au[i])/dT)
    , np.transpose(np.array([x[i+1], y[i+1], th[i+1]]) - np.
    array([ x[i],y[i],th[i]])) )))
Vu.append(np.array([0,0]))

Vu = np.array(Vu)

# separamos por vectores de velocidad [v,w]
v_u = np.array(Vu[:,0])
w_u = np.array(Vu[:,1])

## 1.3
#usaremos la misma matriz A calculada antes, y el mismo delta
    T

Xu0 = np.array([0,0,0]) # vector de coordenadas en k #
    seteamos la primera posicion, en el origen
Xu1 = [] # vector de coordenadas en k + 1

Xu1.append(Xu0) # seteamos la primera posicion
for i in range(x.size-1):
    Xu1.append(Xu0 + dT*( np.matmul(Au[i],np.transpose(np.
    array([v_u[i],w_u[i]])))))
    Xu0 = Xu1[-1]

Xu1 = np.array(Xu1)

xu_new = []
yu_new = []
thu_new = []

xu_new = np.array(Xu1[:,0])
yu_new = np.array(Xu1[:,1])
thu_new = np.array(Xu1[:,2])

## animaci n

x_data = []
y_data = []

fig, ax = plt.subplots()
ax.set_xlim(-10,D+10)
ax.set_ylim(-10,D+10)
line, = ax.plot(0,0)

def animation_frame(i):
    x_data.append(xu_new[i])
    y_data.append(yu_new[i])

    line.set_xdata(x_data)
    line.set_ydata(y_data)
    return line,

animation = FuncAnimation(fig, func=animation_frame, frames=np
    .arange(x.size),interval=dT)
plt.show()

plt.plot(xu_new, yu_new, '.', color='blue')
plt.show()

```

```
219 v_time = [] # tiempo para graficar las velocidades
220 ## grafico de la se al de control vs tiempo
221 for i in range(x.size):
222     v_time.append(dT*i)
223 v_time = np.array(v_time)
224
225 plt.plot(v_time,V1, '.', color='blue',label="v1")
226 plt.plot(v_time,V2, '.', color='green',label="v2")
227 plt.plot(v_time,V3, '.', color='red',label="v3")
228 plt.xlabel("Tiempo [s]")
229 plt.ylabel("Velocidad [m/s]")
230 plt.legend()
231 plt.show()
```

4.2. Pregunta 2

4.2.1. Movimiento de Car-Like Robot usando el teclado

```
import sim
import numpy as np
import keyboard

#funcion necesaria para establecer conexion con coppeliasim
def connect(port):
    sim.simxFinish(-1)
    clientID=sim.simxStart('127.0.0.1', port, True, True,
    2000, 5)
    if clientID == 0: print("conectado a", port)
    else: print("no se pudo conectar")
    return clientID

#establecemos la conexion con coppeliasim
clientID = connect(19999)
#obtener el handler para el car-like
returnCode, car_like = sim.simxGetObjectHandle(clientID, '
    Manta', sim.simx_opmode_blocking)

returnCode, steer_handle = sim.simxGetObjectHandle(clientID,'
    steer_joint', sim.simx_opmode_blocking)
returnCode, motor_handle = sim.simxGetObjectHandle(clientID,'
    motor_joint', sim.simx_opmode_blocking)
returnCode, fl_brake_handle = sim.simxGetObjectHandle(clientID
    ,'fl_brake_joint', sim.simx_opmode_blocking)
returnCode, fr_brake_handle = sim.simxGetObjectHandle(clientID
    ,'fr_brake_joint', sim.simx_opmode_blocking)
returnCode, bl_brake_handle = sim.simxGetObjectHandle(clientID
    ,'bl_brake_joint', sim.simx_opmode_blocking)
returnCode, br_brake_handle = sim.simxGetObjectHandle(clientID
    ,'br_brake_joint', sim.simx_opmode_blocking)

max_steer_angle=0.5235987
motor_torque=60
dVel=1
dSteer=0.1
steer_angle=0
motor_velocity=0#dVel*10
brake_force=0

while(1):  # making a loop
    if(keyboard.is_pressed('w')):  # Control del teclado para
    mover usando 'w', 'a', 's' 'd' para direcciones y ' ' para
    frenar.
        if (motor_velocity<dVel*9.99):
            motor_velocity=motor_velocity+dVel
    if keyboard.is_pressed('s'):
        if (motor_velocity>-dVel*4.99):
            motor_velocity=motor_velocity-dVel
        else:
            brake_force=100
    if(keyboard.is_pressed('a')):
        if (steer_angle<dSteer*4.99):
            steer_angle=steer_angle+dSteer
    if( keyboard.is_pressed('d')):
```

```
46          if (steer_angle>-dSteer*4.99):
47              steer_angle=steer_angle-dSteer
48      if(keyboard.is_pressed(' ')):# freno
49          if(motor_velocity>0):
50              motor_velocity -= 0.01
51          elif (motor_velocity<0):
52              motor_velocity += 0.01
53      else:
54          if(np.abs(motor_velocity)>0):
55              if(motor_velocity>0):# si no se apreta ninguna
    tecla, se frena solo
56                  motor_velocity -= 0.001
57              elif (motor_velocity<0):
58                  motor_velocity += 0.001
59      if (np.abs(motor_velocity)<dVel*0.1):
60          brake_force=100
61      else:
62          brake_force=0
63      #--set maximum steer angle
64      if (steer_angle > max_steer_angle):
65          steer_angle = max_steer_angle
66      if (steer_angle < -max_steer_angle):
67          steer_angle = -max_steer_angle
68      sim.simxSetJointTargetPosition(clientID, steer_handle,
    steer_angle, sim.simx_opmode_oneshot )
69      #--brake and motor can not be applied at the same time
70      if(brake_force>0):
71          sim.simxSetJointMaxForce(clientID, motor_handle, 0,
    sim.simx_opmode_oneshot)
72      else:
73          sim.simxSetJointMaxForce(clientID, motor_handle,
    motor_torque, sim.simx_opmode_oneshot)
74          sim.simxSetJointTargetVelocity(clientID, motor_handle,
     motor_velocity, sim.simx_opmode_oneshot)
75
76      sim.simxSetJointMaxForce(clientID, fr_brake_handle,
    brake_force, sim.simx_opmode_oneshot)
77      sim.simxSetJointMaxForce(clientID, fl_brake_handle,
    brake_force, sim.simx_opmode_oneshot)
78      sim.simxSetJointMaxForce(clientID, br_brake_handle,
    brake_force, sim.simx_opmode_oneshot)
79      sim.simxSetJointMaxForce(clientID, bl_brake_handle,
    brake_force, sim.simx_opmode_oneshot)
```

4.2.2. Manta Child script

```
function sysCall_init()
    -- do some initialization here
    simRemoteApi.start(19999)
end

function sysCall_nonSimulation()
    -- is executed when simulation is not running
end

function sysCall_beforeSimulation()
    -- is executed before a simulation starts
end

function sysCall_afterSimulation()
    -- is executed before a simulation ends
end

function sysCall_cleanup()
    -- do some clean-up here
end

-- See the user manual or the available code snippets for
    additional callback functions and details
```

4.2.3. Uso del sensor Hokuyo

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  7 03:48:28 2021

@author: juan_
"""

import sim
import numpy as np
import keyboard
import sys
import time
import matplotlib.pyplot as plt

#funcion necesaria para establecer conexion con coppeliasim
def connect(port):
    sim.simxFinish(-1)
    clientID=sim.simxStart('127.0.0.1', port, True, True,
    2000, 5)
    if clientID == 0: print("conectado a", port)
    else: print("no se pudo conectar")
    return clientID

#establecemos la conexion con coppeliasim
clientID = connect(19999)
#obtener el handler para el car-like
returnCode, car_like = sim.simxGetObjectHandle(clientID, '
    Manta', sim.simx_opmode_blocking)

returnCode, steer_handle = sim.simxGetObjectHandle(clientID,'
    steer_joint', sim.simx_opmode_blocking)
returnCode, motor_handle = sim.simxGetObjectHandle(clientID,'
    motor_joint', sim.simx_opmode_blocking)
returnCode, fl_brake_handle = sim.simxGetObjectHandle(clientID
    ,'fl_brake_joint', sim.simx_opmode_blocking)
returnCode, fr_brake_handle = sim.simxGetObjectHandle(clientID
    ,'fr_brake_joint', sim.simx_opmode_blocking)
returnCode, bl_brake_handle = sim.simxGetObjectHandle(clientID
    ,'bl_brake_joint', sim.simx_opmode_blocking)
returnCode, br_brake_handle = sim.simxGetObjectHandle(clientID
    ,'br_brake_joint', sim.simx_opmode_blocking)

max_steer_angle=0.5235987
motor_torque=60
dVel=1
dSteer=0.1
steer_angle=0
motor_velocity=0#dVel*10
brake_force=0

returnCode, gps_handler = sim.simxGetObjectHandle(clientID,'
    GPS', sim.simx_opmode_blocking) #gps
returnCode, gyro_handler = sim.simxGetObjectHandle(clientID,'
    GyroSensor', sim.simx_opmode_blocking) #gps
errorCode, ranges = sim.simxGetStringSignal(clientID, 'scan
    ranges', sim.simx_opmode_streaming)# sensor
time.sleep(0.1)
```

```
47
48 while(1):  # making a loop
49     time.sleep(0.1)
50     #try:  # used try so that if user pressed other than the
       given key error will not be shown
51     if(keyboard.is_pressed('w')):  # if key 'q' is pressed
52         if (motor_velocity<dVel*9.99):
53             motor_velocity=motor_velocity+dVel
54     if keyboard.is_pressed('s'):
55         if (motor_velocity>-dVel*4.99):
56             motor_velocity=motor_velocity-dVel
57         else:
58             brake_force=100
59     if(keyboard.is_pressed('a')):
60         if (steer_angle<dSteer*4.99):
61             steer_angle=steer_angle+dSteer
62     if( keyboard.is_pressed('d')):
63         if (steer_angle>-dSteer*4.99):
64             steer_angle=steer_angle-dSteer
65     if(keyboard.is_pressed(' ')):# freno
66         if(motor_velocity>0):
67             motor_velocity -= 0.1
68         elif (motor_velocity<0):
69             motor_velocity += 0.01
70     else:
71         if(np.abs(motor_velocity)>0):
72             if(motor_velocity>0):# si no se apreta ninguna
       tecla, se frena solo
73                 motor_velocity -= 0.001
74             elif (motor_velocity<0):
75                 motor_velocity += 0.001
76     if (np.abs(motor_velocity)<dVel*0.1):
77         brake_force=100
78     else:
79         brake_force=0
80     #--set maximum steer angle
81     if (steer_angle > max_steer_angle):
82         steer_angle = max_steer_angle
83     if (steer_angle < -max_steer_angle):
84         steer_angle = -max_steer_angle
85     sim.simxSetJointTargetPosition(clientID, steer_handle,
       steer_angle, sim.simx_opmode_oneshot )
86     #--brake and motor can not be applied at the same time
87     if(brake_force>0):
88         sim.simxSetJointMaxForce(clientID, motor_handle, 0,
       sim.simx_opmode_oneshot)
89     else:
90         sim.simxSetJointMaxForce(clientID, motor_handle,
       motor_torque, sim.simx_opmode_oneshot)
91         sim.simxSetJointTargetVelocity(clientID, motor_handle,
        motor_velocity, sim.simx_opmode_oneshot)
92
93     sim.simxSetJointMaxForce(clientID, fr_brake_handle,
       brake_force, sim.simx_opmode_oneshot)
94     sim.simxSetJointMaxForce(clientID, fl_brake_handle,
       brake_force, sim.simx_opmode_oneshot)
95     sim.simxSetJointMaxForce(clientID, br_brake_handle,
       brake_force, sim.simx_opmode_oneshot)
96     sim.simxSetJointMaxForce(clientID, bl_brake_handle,
       brake_force, sim.simx_opmode_oneshot)
```

```
97
98      #angulo actual con gyroscopio
99      returnCode, gir = sim.simxGetObjectPosition(clientID,
        gyro_handler, -1, sim.simx_opmode_blocking)
100     #posicion actual con gps:
101     returnCode, pos = sim.simxGetObjectPosition(clientID,
        gps_handler, -1, sim.simx_opmode_blocking)
102     # Obtenga datos v lidos
103     errorCode, ranges = sim.simxGetStringSignal(clientID, '
        scan ranges', sim.simx_opmode_buffer)
104     #Convertir cadena a lista flotante, el valor en la lista
        es el valor medido del radar
105     ranges = sim.simxUnpackFloats(ranges)
106     plt.xlim(0, 684)
107     plt.ylim(0, 5)
108     x = range(len(ranges))
109     y = ranges
110     # Dibuja el resultado
111     plt.scatter(x, y)
112     plt.show()
```

4.2.4. Child Script fastHokuyo

```
function sysCall_init()
    local self=sim.getObjectHandle(sim.handle_self)
    visionSensor1Handle=sim.getObjectHandle("fastHokuyo_sensor1")
    visionSensor2Handle=sim.getObjectHandle("fastHokuyo_sensor2")
    joint1Handle=sim.getObjectHandle("fastHokuyo_joint1")
    joint2Handle=sim.getObjectHandle("fastHokuyo_joint2")
    sensorRef=sim.getObjectHandle("fastHokuyo_ref")
    local collection=sim.createCollection(0)
    sim.addItemToCollection(collection,sim.handle_all,-1,0)
    sim.addItemToCollection(collection,sim.handle_tree,self,1)
    sim.setObjectInt32Param(visionSensor1Handle,sim.visionintparam_entity_to_render,collection)
    sim.setObjectInt32Param(visionSensor2Handle,sim.visionintparam_entity_to_render,collection)


    maxScanDistance=5
    sim.setObjectFloatParam(visionSensor1Handle,sim.visionfloatparam_far_clipping,maxScanDistance)
    sim.setObjectFloatParam(visionSensor2Handle,sim.visionfloatparam_far_clipping,maxScanDistance)
    maxScanDistance_=maxScanDistance*0.9999

    scanningAngle=240*math.pi/180
    sim.setObjectFloatParam(visionSensor1Handle,sim.visionfloatparam_perspective_angle,scanningAngle/2)
    sim.setObjectFloatParam(visionSensor2Handle,sim.visionfloatparam_perspective_angle,scanningAngle/2)

    sim.setJointPosition(joint1Handle,-scanningAngle/4)
    sim.setJointPosition(joint2Handle,scanningAngle/4)
    red={1,0,0}
    lines=sim.addDrawingObject(sim.drawing_lines,1,0,-1,10000,nil,nil,nil,red)
    showLines=true
end

function sysCall_cleanup()
    sim.removeDrawingObject(lines)
end

function sysCall_sensing()
    measuredData={}
    ranges = {}
    ------------------------------------------------------added
    if notFirstHere then
        -- We skip the very first reading
        sim.addDrawingObjectItem(lines,nil)
        r,t1,u1=sim.readVisionSensor(visionSensor1Handle)
        r,t2,u2=sim.readVisionSensor(visionSensor2Handle)

        m1=sim.getObjectMatrix(visionSensor1Handle,-1)

        m01=sim.getObjectMatrix(sensorRef,-1)
        sim.invertMatrix(m01)
```

```
48          m01=sim.multiplyMatrices(m01,m1)
49          m2=sim.getObjectMatrix(visionSensor2Handle,-1)
50          m02=sim.getObjectMatrix(sensorRef,-1)
51          sim.invertMatrix(m02)
52          m02=sim.multiplyMatrices(m02,m2)
53          if u1 then
54              p={0,0,0}
55              p=sim.multiplyVector(m1,p)
56              t={p[1],p[2],p[3],0,0,0}
57              for j=0,u1[2]-1,1 do
58                  for i=0,u1[1]-1,1 do
59                      w=2+4*(j*u1[1]+i)
60                      v1=u1[w+1]
61                      v2=u1[w+2]
62                      v3=u1[w+3]
63                      v4=u1[w+4]
64                      if (v4<maxScanDistance_) then
65                          p={v1,v2,v3}
66                          p=sim.multiplyVector(m01,p)
67                          table.insert(measuredData,p[1])
68                          table.insert(measuredData,p[2])
69                          table.insert(measuredData,p[3])
70                          table.insert(ranges, v4)
------------------added
71                      else
-------------------------------------------added
72                          table.insert(ranges, 0)
--------------------added
73                      end
74                      if showLines then
75                          p={v1,v2,v3}
76                          p=sim.multiplyVector(m1,p)
77                          t[4]=p[1]
78                          t[5]=p[2]
79                          t[6]=p[3]
80                          sim.addDrawingObjectItem(lines,t)
81                      end
82                  end
83              end
84          end
85          if u2 then
86              p={0,0,0}
87              p=sim.multiplyVector(m2,p)
88              t={p[1],p[2],p[3],0,0,0}
89              for j=0,u2[2]-1,1 do
90                  for i=0,u2[1]-1,1 do
91                      w=2+4*(j*u2[1]+i)
92                      v1=u2[w+1]
93                      v2=u2[w+2]
94                      v3=u2[w+3]
95                      v4=u2[w+4]
96                      if (v4<maxScanDistance_) then
97                          p={v1,v2,v3}
98                          p=sim.multiplyVector(m02,p)
99                          table.insert(measuredData,p[1])
100                         table.insert(measuredData,p[2])
101                         table.insert(measuredData,p[3])
102                         table.insert(ranges, v4)
------------------added
```

```
103                     else
    ------------------------------------------added
104                         table.insert(ranges, 0)
    --------------------added
105                     end
106                     if showLines then
107                         p={v1,v2,v3}
108                         p=sim.multiplyVector(m2,p)
109                         t[4]=p[1]
110                         t[5]=p[2]
111                         t[6]=p[3]
112                         sim.addDrawingObjectItem(lines,t)
113                     end
114                 end
115             end
116         end
117         ranges = sim.packFloatTable(ranges)
    ------------------------addedd
118         sim.setStringSignal('scan ranges', ranges)
    ----------------addedd
119     end
120     notFirstHere=true
121 end
```